Apple Assembly Line

$1.80

Volume 6 -- Issue 3 December, 1985

In This Issue...

Little RAM Disk Bug

Does that mean a Bug in the Little RAM Disk, or a Little Bug in the RAM Disk? Actually, both. Several of you have called or written to point out a problem in Bob's program last month

The TAY instruction at line 1280 (on page 8) should be a TYA. It does seem pointless to force Y to zero and then immediately clobber it with whatever the processor read out of $C083. This code worked when Bob tested it on his //e, because that computer does return a zero when you read $C083. My][+, on the other hand, returns a byte of video data, usually $A0, and that really makes a mess out of the VTOC and Catalog sectors.

There's one other glitch in that article as well. In the fifth paragraph on page 5 there is a reference to line number "189~". I bet you can guess that's really supposed to be "1890".

So, thanks to all of you who caught us on this one! It's nice to know you're keeping an eye on us.

ProDOS MLI Tracing..................................Ken Kashmarek
Eldridge, Iowa

I took Bob S-C's work with ProDOS Snooper (October 1985 AAL) one step further: I added MLI calls to the information that is collected in the trace table. By combining the MLI call data with the device driver data, we get a better idea of what is happening.

The entries below all come from slot 6 drive 1. MLI calls are tagged with an "M" after the hex data. To support both the MLI calls and device driver calls, the hex output provides the data as it exists in memory without taking into account whether a set of bytes is a two byte memory pointer or a single data byte.

For all calls, the return address is still shown as hi-byte first before the colon. Data for the device driver parameter is still from $42-$47. For MLI calls, the return address is to the program that called the routine in the BASIC.SYSTEM global page. All BASIC.SYSTEM calls go to the $BE00 global page and then to the $BF00 ProDOS global page. MLI data is the MLI call number followed by the first five bytes of the parameter list (some bytes do not apply if the list is shorter).

The volume in question is labeled /TEST and has one file, ABC, in the root directory.

First of all, issue: CAT,S6

```
      A6E9:C7 BC BC 02 BC BC M GET PREFIX
      A85F:C5 60 01 02 00 03 M ON LINE CALL     + Not used when
      EC0C:01 50 00 DC 02 00   READ BLOCK 2     + CAT /TEST
      entered
      A825:C4 BC BC C3 0F 00 M GET FILE INFO
      EC0C:01 60 00 DC 02 00   READ BLOCK 2
      EC0C:01 60 00 DC 06 00   READ Bit Map
      B1B9:C8 BC BC 00 8A 01 M OPEN FILE
      EC0C:01 60 00 DC 02 00   READ BLOCK 2
      EE85:01 60 00 8A 02 00   READ BLOCK 2
      B175:CA 01 59 02 2B 00 M READ FILE
      B201:CE 01 2B 00 00 03 M SET FILE MARK    + Appears for each
      B208:CA 01 59 02 27 00 M READ FILE        + file in
      directory
      B0A5:CC 01 00 C3 CF D0 M CLOSE FILE
      B0FB:C5 60 BD BC 00 03 M ON LINE CALL
      EC0C:01 50 00 DC 02 00   READ BLOCK 2
      B10F:C4 BC BC C3 0F 18 M GET FILE INFO
      EC0C:01 60 00 DC 02 00   READ BLOCK 2
      EC0C:01 60 00 DC 06 00   READ Bit Map
```

For this simple operation, there are ten MLI calls and eight device driver calls (disk I/O operations). I do not understand the reason for the Get Prefix call at the beginning. It would appear that the On Line call and the Get File Info call at the end are unnecessary (we will be checking this out as we go). On Line returns the volume name, but this should already be available through the prefix or pathname of the directory. Get

```
S-C Macro Assembler Version 2.0......DOS $100, ProDOS $100, both for $120
ProDOS Upgrade Kit for Version 2.0 DOS owners..........................$30
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners.........................$20
Source Code of S-C Macro 2.0 (DOS only)..................additional $100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility......without source code $20, with source $50
RAK-Ware DISASM........................................................$30
Source Code for DISASM...................................additional $30
S-C Word Processor (with complete source code).........................$50
DP18 Source and Object.................................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95)  $36 *
MacASM -- Macro Assembler for MacIntosh (Mainstay)..........($150.00) $100 *
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk................................$15
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
   (Available:  6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048,
                8051, 8085, 1802/4/5, PDP-11, GI1650/70, others)

AAL Quarterly Disks.........................each $15, or any four for $45
 Each disk contains the source code from three issues of AAL,
 saving you lots of typing and testing.
 The quarters are Jan-Mar, Apr-Jun, Jul-Sep, and Oct-Dec.
(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

Diskettes (with hub rings)......................... package of 20 for $32 *
Vinyl disk pages, 6"x8.5", hold two disks each.................10 for $6 *
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100 *
Envelopes for Diskette Mailers............................  6 cents each

65802 Microprocessor (Western Design Center)..................($95)  $50 *
quikLoader EPROM System (SCRG)...............................($179) $170 *
PROmGRAMER (SCRG)..........................................($149.50) $140 *
Switch-a-Slot (SCRG).......................................($179.50) $170 *
Extend-a-Slot (SCRG)..........................................($35)  $32 *
Kablit Security System (Secure-It, Inc.)....................($49.95)  $45 *

"Programming the 65816", Eyes...............................($22.95)  $21 *
"Apple //e Reference Manual", Apple Computer................($24.95)  $23 *
"Apple //c Reference Manual", Apple Computer................($24.95)  $23 *
"ProDOS Technical Reference Manual", Apple Computer.........($29.95)  $27 *
"Now That You Know Apple Assembly Language...", Gilder......($19.95)  $18 *
"Apple ProDOS:  Advanced Features for Programmers", Little..($17.95)  $17 *
"Inside the Apple //c", Little..............................($19.95)  $18 *
"Inside the Apple //e", Little..............................($19.95)  $18 *
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)  $18 *
"Apple ][ Circuit Description", Gayler......................($22.95)  $21 *
"Understanding the Apple II", Sather........................($22.95)  $21 *
"Understanding the Apple //e", Sather.......................($24.95)  $23 *
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95)  $15 *
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95)  $17 *
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)  $20 *
"Beneath Apple DOS", Worth & Lechner........................($19.95)  $18 *
"Beneath Apple ProDOS", Worth & Lechner.....................($19.95)  $18 *
"Real Time Programming -- Neglected Topics", Foster..........($9.95)   $9 *
"Microcomputer Graphics", Myers.............................($12.95)  $12 *
"Assem. Language for Applesoft Programmers", Finley & Myers.($16.95)  $16 *
"Assembly Lines -- the Book", Wagner........................($19.95)  $18 *
"AppleVisions", Bishop & Grossberger........................($39.95)  $36 *

           * On these items add $2.00 for the first item and
             $.75 for each additional item for US shipping.
             Foreign customers inquire for postage needed.
        Texas residents please add 6 1/8 % sales tax to all orders.

      *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
      ***                  (214) 324-2050                   ***
      *** We accept Master Card, VISA and American Express ***
```

File Info information should already be available from the previous call, and the bit map was already read in once. However, this is a simple catalog operation and may be indicative of some of the steps necessary for more complex catalog operations.

Carrying this one step further, I issued CAT /TEST/DIR. In this case, the first read of the bit map is not performed. Next, the former apparently duplicate read of block 2 now turns into a read of block 7, the key block for subdirectory DIR (in /TEXT/DIR; the device driver return address is $EE85, the buffer address is $8A00). Note: block 2 is the key block of the root directory.

A Get File Info call for a volume name (/TEST) always reads the bit map. Therefore, this call is repeated when cataloging a volume, but not when cataloging a subdirectory. As to the On Line call, it is used to get volume name for the Get File Info call for the free space information for the volume, since the initial catalog command may have been for a subdirectory. This explains (only partially) what appeared to be duplicate reads of the same information.

Now, let's try loading an Applesoft file: LOAD ABC,S6

```
A85F:C5 60 01 02 00 03 M ON LINE CALL    + Not used for
EC0C:01 60 00 DC 02 00   READ BLOCK 2    + LOAD /TEST/ABC
A825:C4 BC BC E3 FC 01 M GET FILE INFO
EC0C:01 60 00 DC 02 00   READ BLOCK 2
AC00:CC 00 00 C3 CF D0 M CLOSE ALL FILES
B1B9:C8 BC BC 00 8A 01 M OPEN FILE
EC0C:01 60 00 DC 02 00   READ BLOCK 2
EE85:01 60 00 8A 07 00   READ BLOCK 7
AC22:D1 01 01 02 00 03 M GET FILE EOF
AC4B:CA 01 01 08 09 00 M READ FILE
AC50:CC 01 00 C3 CF D0 M CLOSE FILE
```

The loaded program is less than 512 bytes in length, so the key block read is the only data I/O operation. As with the catalog operation, the Get File Info call is used to verify the file type. Close All Files is used in case the previous program left any open. Note the Get File EOF call which is used to get the length for the Read File call (which performs the entire load operation). This example is relatively simple. Let's check what happens when we create an Applesoft file that is just over 512 bytes in length (changing our seedling file into a sapling file, which requires an index block and two data blocks).

We'll lengthen the program, and then type: SAVE /TEST/ABC.3

```
A825:C4 BC BC C3 0F 18 M GET FILE INFO
EC0C:01 60 00 DC 02 00   READ BLOCK 2
ACDC:C0 BC BC C3 FC 01 M CREATE FILE
EC0C:01 60 00 DC 02 00   READ BLOCK 2
F477:00 60 00 DC 00 00   STATUS S6.D1
EC0C:01 60 00 DA 06 00   READ BIT MAP
EC0C:02 60 00 DC 07 00   WRITE BLOCK 7
```

```
ECOC:01 60 00 DC 02 00   READ BLOCK 2
ECOC:02 60 00 DC 02 00   WRITE BLOCK 2
ECOC:02 60 00 DA 06 00   WRITE BIT MAP
B1B9:C8 BC BC 00 8A 01 M OPEN FILE CALL
ECOC:01 60 00 DC 02 00   READ BLOCK 2
EE85:01 60 00 8A 07 00   READ BLOCK 7
AD0A:CB 01 01 08 5B 02 M WRITE FILE CALL
F477:00 60 01 08 00 00   STATUS S6,D1
EE85:02 60 00 8A 07 00   WRITE BLOCK 7
ECOC:01 60 00 DA 06 00   READ BIT MAP
ECOC:02 60 00 DA 06 00   WRITE BIT MAP
EE85:02 60 00 8C 08 00   WRITE BLOCK 8
ECOC:01 60 00 DA 06 00   READ BIT MAP
AD11:D0 01 5B 02 00 03 M SET FILE EOF CALL
AD16:CC 01 00 C3 CF D0 M CLOSE FILE CALL
EE85:02 60 00 8A 09 00   WRITE BLOCK 9
ECOC:02 60 00 DA 06 00   WRITE BIT MAP
EE85:02 60 00 8C 08 00   WRITE BLOCK 8
ECOC:01 60 00 DC 02 00   READ BLOCK 2
ECOC:01 60 00 DC 02 00   READ BLOCK 2
ECOC:02 60 00 DC 02 00   WRITE BLOCK 2
```

This sequence has the same number of MLI calls for a seedling or a sapling file. The big difference is allocating the index block (block number 8) and additional data blocks. This also generates additional calls to read and write the bit map.

If the file already exists, and the SAVE command does not change the length, then the Create File call is not executed, there are no accesses to the bit map (block 6), and the index block does not change. If the file length changes sufficiently to add or delete blocks, then the bit map is updated and the index block is rewritten (this is forced by the Set File EOF call which adjusts the file length).

Interesting note: whenever a file is opened, the first data block is always read in, even if the file will subsequently be written to. Likewise, when a new file is allocated, the first data block is allocated and written, even if no data is placed in the block.

In the above sequence, what appears to be a duplicate read of block 2 (return address $ECOC) is actually a read to separate blocks if the SAVE command was to a subdirectory. It turns out to be duplicate reads to the subdirectory block, write to the subdirectory, then read and write the root directory. Sigh.

LOAD /TEST/ABC.3 is similar to the previous load operation, except that we must also read the index block before reading the data blocks, and there are two data blocks rather than one.

Finally, let's try deleting this file: DELETE /TEST/ABC.3

```
A825:C4 BC BC E3 04 00 M GET FILE INFO CALL
ECOC:01 60 00 DC 02 00   READ BLOCK 2
9AD7:C1 BC BC 02 BC BC M DESTROY FILE CALL
ECOC:01 60 00 DC 02 00   READ BLOCK 2
F477:00 60 00 DC 00 00   STATUS S6,D1
```

```
ECOC:01 60 00 DC 08 00    READ BLOCK 8 (index block)
ECOC:01 60 00 DA 06 00    READ BIT MAP
ECOC:02 60 00 DC 08 00    WRITE INDEX BLOCK (zeroed)
ECOC:01 60 00 DC 07 00    READ BLOCK 7
ECOC:02 60 00 DA 06 00    WRITE BIT MAP
ECOC:01 60 00 DC 02 00    READ BLOCK 2
ECOC:02 60 00 DC 02 00    WRITE BLOCK 2
```

Again, use Get File Info for file type and status call to see if the disk can be written to. The bit map is read and written to reflect the freed blocks. Block 8, the former file index block, is trashed. I don't know why block 7 is read in. Trashing the index block makes it very hard to reconstruct a DELETEd file.

At this point, we get a feel for what is happening between the MLI calls and the device driver calls. Consider how extensive these simple examples become on a hard disk if working down three or four directory levels and at the second, third, or fourth block in each directroy, and the hard disk has five blocks for the bit map (and we need the fifth block because the disk is almost full). Ouch!

I performed one more test case, far too long to list here. It involved adding a record to a new sparse random access file. The new record caused the file to grow to a tree file. The program used was:

```
10 D$ - CHR$(4)
20 PRINT D$"OPEN /TEST/NAMES,L140"
30 PRINT D$"WRITE/TEST/NAMES,R936"
40 PRINT "XXX ... XXX": REM 120 X's
50 PRINT D$"CLOSE/TEST/NAMES"
```

This sequence produced eight MLI calls and 29 device driver calls to perform I/O (there were three status calls). The file ended up with six blocks (master index block, two index blocks, and three data blocks) which generated 12 accesses to read and write the bit map.

A 32 megabyte hard disk, the maximum size supported by ProDOS, requires 16 blocks for the free space bit map. Obviously, such a disk would suffer quite a performance impact when allocating new files, or adding space to existing files, if the hard disk were more than half full.

Ohio Systems Kache Card....................Bob Sander-Cederlof

After reading Ken's article, I came to the conclusion that the Kache Card or something like it is a MUST for users of large hard disks.

The Kache Card has 256K RAM and a controlling Z-80 on it. As far as the Apple is concerned, it is just a hard disk controller. It replaces the controller card which came with your Sider. But it is smarter.

The Kache card remembers the most recently read or most frequently read data blocks. Over 2000 of them. You can see that the entire bit map and at least all the directory blocks associated with the currently used pathnames would stay in RAM on the card. When ProDOS issues a READ command, the DMA interface on the Kache Card simply transfers the block, without doing anything to the hard disk.

When you write to the hard disk, the Kache Card sends it to the hard disk as well as updating its RAM-based copy. You can write to the Kache Card faster than the Kache Card writes to the disk, and your program keeps chugging along while the Kache Card spins out the data to the drive.

The Kache Card is expensive ($695), at least relative to the price of a Sider. A 10-meg Sider is currently $595, and a 20-meg Sider is currently $895. Nevertheless, if you are using 20 megabytes or more you really need a caching system of some kind.

Of course, you could implement caching inside the operating system. ProDOS could be modified (perish the thought) to use about 16K RAM from the //e's auxiliary memory to cache the bit map, root directory, and other frequently used blocks, for each on-line hard disk. (It does not seem profitable to try to cache blocks from floppies, because you can too easily mess things up by removing one floppy and inserting another.)

Like I said, you COULD do it this way. However, it would be very difficult to make it work with the variety of peripherals available to Apple owners. It seems much more reasonable to include caching on the controller card, or even inside the hard disk box itself. I think 256K is probably overkill, 64K per hard drive should be plenty.

My first brush with the Kache Card was not pleasant. I ended up returning the card with a list of complaints. They called me about a month later with the news that they had taken my compaints seriously, and rectified the problems I had pointed out. Or at least most of them.

If you are interested in the Kache Card, contact Ohio Kache Systems Corporation, 75 Tahlequah Trail, Springboro, Ohio 45066. Or call them at (513) 746-9160. Tell them where you read this.

More Puzzle Solutions..............Bruce Love & Charles Putney

It takes a little longer for the mail to carry our messages overseas, so these solutions missed the November issue.

Bruce Love (from Hamilton, New Zealand) uses the power of the 65802 in a different manner than David Johnson did last month. Remember that David used the MVP instruction to fill all RAM with the STP opcode. Bruce uses a combination of a loop and the PHA instruction to fill all of RAM with $4C, which is a JMP opcode.

If you disassemble a series of $4C's, you will see JMP $4C4C. Therefore Bruce positioned his code so that the last byte to be filled is at $4C4C.

The loop in lines 1160-1200 fills all RAM below $4C4C with the $4C value. After finishing, it jumps back to $4C4C where a two-line loop pushes the A-register on the stack. The trick here is that the stack pointer in the 65802 is 16-bits long. Bruce starts it at $BFFF, and each PHA lowers it by one location. The last location to be changed is $4C4C itself, and after that it loops endlessly executing JMP $4C4C at $4C4C.

Bruce points out that you can test the effectiveness of his program (if you have a 65802 in your Apple) by changing lines 1130 and 1160 to LDX ##$4FFF and LDX ##$4000 respectively. Then it will fill the range from $4000 through $4FFF with $4C, and you can examine it to be sure it did.

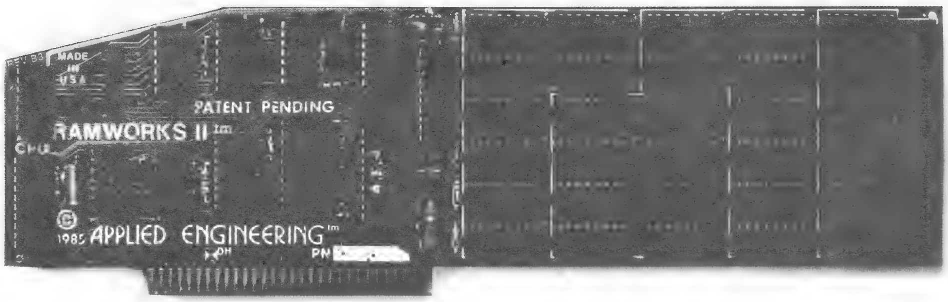
Charles Putney (from Shankill, Dublin, Ireland) fills RAM with $48, using the normal 6502 instruction set. Charlie used a combination similar to Bob S-C's solution last month. The final loop resides inside the stack page, and the infinite series of PHA's fills the stack. The difference is that Charlie has the user type an "L" key, which loads the keyboard register with $CC. Then he clears the strobe, which changes it to $4C. Since the locations $C000 through $C002 will all read back as $4C, the cpu will execute JMP $4C4C.

```
                  1000 *SAVE S.RAMFILL BRUCE LOVE
                  1010 *--------------------------------
                  1020        .OP 65802
                  1030        .OR $4C49
                  1040 *--------------------------------
004C49- 4C 50 4C  1050 PAINT  JMP .2
                  1060 *--------------------------------
004C4C- 48        1070 .1     PHA          PUSH FROM $BFFF DOWN
004C4D- 4C 4C 4C  1080        JMP .1       (NOTE = 4C4C4C)
                  1090 *--------------------------------
004C50- 18        1100 .2     CLC          TURN ON 65802 MODE
004C51- FB        1110        XCE
004C52- C2 10     1120        REP #$10     X=16 BIT, A=8 BIT
004C54- A2 FF BF  1130        LDX ##$BFFF  POINT STACK TO TOP OF RAM
004C57- 9A        1140        TXS
004C58- A9 4C     1150        LDA #$4C     FILL VALUE
004C5A- A2 00 00  1160        LDX ##0      POINT TO BOTTOM OF RAM
004C5D- 9D 00 00  1170 .3     STA >0,X     FILL FROM $0000 TO $4C4B
004C60- E8        1180        INX
004C61- E0 4C 4C  1190        CPX ##$4C4C
004C64- 90 F7     1200        BCC .3
004C66- B0 E4     1210        BCS .1       BACK TO FILL FROM TOP DOWN
                  1220 *--------------------------------
```

```
                  1000 *SAVE S.RAMFILL PUTNEY
                  1010 *-------------------------------
                  1020 *  BY   CHARLES H. PUTNEY
                  1030 *       18 QUINNS ROAD
                  1040 *       SHANKILL
                  1050 *       CO. DUBLIN
                  1060 *       IRELAND
                  1070 *-------------------------------
                  1080        .OR $803      NORMAL PLACE
                  1090 *-------------------------------
06-               1100 PNTR   .EQ $06       BLOCK MOVE POINTER
                  1110 *-------------------------------
C000-             1120 KEYBD  .EQ $C000     KEYBOARD DATA
C010-             1130 KEYSTB .EQ $C010     KEYBOARD STROBE
FDF0-             1140 VIDOUT .EQ $FDF0     VIDEO OUTPUT ROUTINE
FD8E-             1150 CROUT  .EQ $FD8E     SEND A RETURN
                  1160 *-------------------------------
0803- 20 8E FD    1170 WIPE   JSR CROUT     START A NEW LINE
0806- A2 00       1180        LDX #$00
0808- BD 53 08    1190 .1     LDA MESS,X    TELL HIM WHAT KEY TO PUSH
080B- 20 F0 FD    1200        JSR VIDOUT    SEND IT
080E- E8          1210        INX           NEXT CHAR
080F- A8          1220        TAY           CHECK IF LAST ONE
0810- 30 F6       1230        BMI .1        NO
0812- 20 8E FD    1240        JSR CROUT     SEND A RETURN
0815- A2 00       1250        LDX #$00
0817- BD 34 08    1260 .2     LDA IMAGE,X   RELOCATE CODE TO PAGE ONE
081A- 9D E1 01    1270        STA $200-CODEND+CODE,X
081D- E8          1280        INX
081E- E0 1F       1290        CPX #CODEND-CODE
0820- D0 F5       1300        BNE .2
0822- 2C 00 C0    1310 .3     BIT KEYBD     KEY PRESSED ?
0825- 10 FB       1320        BPL .3        WAIT UNTIL PUSHED ?
0827- AD 10 C0    1330        LDA KEYSTB    RESET STROBE
082A- AD 00 C0    1340        LDA KEYBD     MAKE SURE ITS THE RIGHT KEY
082D- C9 4C       1350        CMP #$4C      IS IT L ? (JMP OPCODE)
082F- D0 D2       1360        BNE WIPE      TELL HIM AGAIN
0831- 4C E1 01    1370        JMP CODE      WIPE OUT !
                  1380 *-------------------------------
                  1390 *   THIS CODE IS RELOCATED TO PAGE ONE
                  1400 *-------------------------------
                  1410 IMAGE  .PH $1E1
01E1- A9 00       1420 CODE   LDA #$00      INITIALIZE POINTER
01E3- 85 06       1430        STA PNTR
01E5- A9 02       1440        LDA #$02
01E7- 85 07       1450        STA PNTR+1    START AT PAGE TWO
01E9- A9 48       1460        LDA #$48      GET A PHA OPCODE
01EB- A0 00       1470        LDY #$00      INIT Y REG
01ED- 91 06       1480 .1     STA (PNTR),Y  SAVE PHA OPCODE
01EF- C8          1490        INY           NEXT
01F0- D0 FB       1500        BNE .1        FULL PAGE DONE ?
01F2- E6 07       1510        INC PNTR+1    NEXT PAGE
01F4- A6 07       1520        LDX PNTR+1    CHECK IF DONE
01F6- E0 C0       1530        CPX #$C0      AT I/O AREA ?
01F8- D0 F3       1540        BNE .1        NOT YET
01FA- 99 00 00    1550 .2     STA $00,Y     SET PAGE ZERO TO $48
01FD- C8          1560        INY           NEXT
01FE- D0 FA       1570        BNE .2        FULL PAGE WIPED ?
                  1580 *   FALL INTO PAGE 2 PHA'S
                  1590 CODEND .EP
                  1600 *-------------------------------
0853- D4 D9 D0
0856- C5 A0 D5
0859- D0 D0 C5
085C- D2 A0 C3
085F- C1 D3 C5
0862- A0 CC A0
0865- D4 CF A0
0868- D3 C5 D4
086B- A0 CD C5
086E- CD CF D2
0871- D9 A0 D4
0874- CF A0 A4
0877- B4 B8 A0
087A- 20          1610 MESS   .AT -/TYPE UPPER CASE L TO SET MEMORY TO $48  /
                  1620 *-------------------------------
```

Meet RamWorks II®

The Recognized Industry Standard For Memory Expansion of the Apple IIe.



RamWorks II. A Generation Ahead. Again.

The best selling expansion card for the Apple IIe just got even better. With RamWorks II, expand your IIe to an incredible 3 megabytes of usable RAM.

Turbo Charged AppleWorks.

RamWorks II plugs into the IIe auxiliary slot and acts just like Apple's extended 80 column card, only better—because if you buy a 256K or larger card, AppleWorks will automatically load itself into RamWorks II. This dramatically increases AppleWorks' speed and power because it effectively eliminates the time required to access disk drive 1. Now, switch from word processing to spreadsheet to database management at the speed of light. AppleWorks responds the moment your fingers touch the keyboard.

But AppleWorks has certain internal limits, independent of available memory. Fear not. Only RamWorks II (and the original RamWorks of course) removes those limits. Only RamWorks II increases the maximum number of records available from 1,350 to over 16 000. Only RamWorks II actually increases the number of lines permitted in the word processing mode. And only RamWorks II features a built-in printer buffer, so you no longer have to wait for your printer to stop before going back to AppleWorks (256K or larger RamWorks II required).

With RamWorks II, you won't have to split your data into 2 or more separate files because you'll have the necessary memory to access ALL your data ALL the time, quickly and conveniently.

RamWorks II	AppleWorks Desktop
128K	101K
256K	188K
512K	378K
1 MEG	758K
1.5 MEG	1136K
3 MEG	2277K

The Most Friendly, Most Expandable Card Available.

RamWorks II is compatible with more off-the-shelf software than any other RAM card. Popular programs like Advanced VisiCalc, Magic Office System, Flashcalc, The Spread Sheet, Diversi-DOS, Supercalc 3A, Magicalc, etc. (and hardware add-ons like Profile and Sider hard disks). Fact is, only RamWorks is 100% compatible with all software written for the Apple 80 column and extended 80 column cards. In addition, RamWorks II can emulate most other RAM cards, so you can use programs written for them without modification. And any size RamWorks II can be user upgraded later to any larger size.

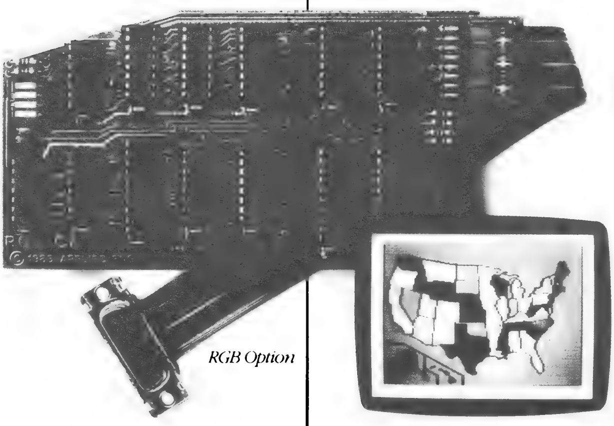
RamWorks II was designed so you could take full advantage of future developments in 16 and 32 bit microprocessors. As your needs grow, so can RamWorks II. A handy coprocessor connector allows the latest and greatest coprocessor cards to access all 3 MEG

of RamWorks II memory. And speaking of more memory, RamWorks II has a memory expansion connector on board so a low profile (no slot 1 interference) memory expansion card can add another 512K or 2 MEG of memory.

Unlike Apple's smaller, more expensive RAM card, RamWorks II plugs into the IIe auxiliary slot and therefore leaves slots 4 and 5 available for other peripheral cards.

It's In Color

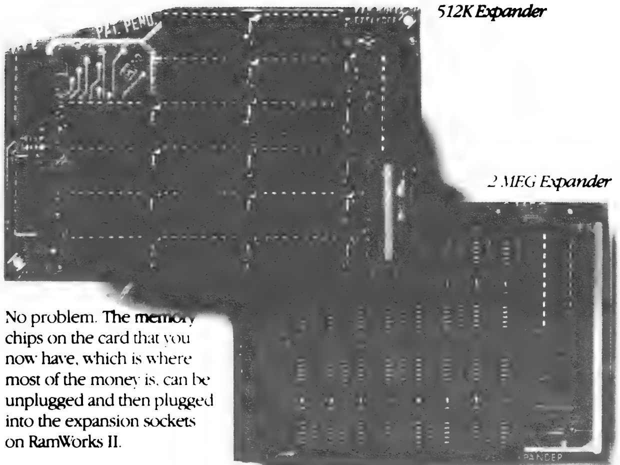
RamWorks II by itself is *fully* compatible with both the Apple monochrome and color monitors. But if you want better color graphics *plus* a more readable 80 column text (that blows away any composite color monitor) you'll appreciate our RGB color option. For only $129, it can be added to RamWorks II, giving you a razor sharp, vivid brilliance that's unsurpassed in the industry. The RGB option does not waste another valuable slot, but rather plugs into the back of RamWorks II with no slot 1 interference (works on the original RamWorks, too) and attaches to virtually any RGB monitor. And remember: You can order the RGB option with your RamWorks II. Or add it on at a later date.



RGB Option

It Corrects Mistakes.

Let's say you bought some other RAM card (and that's a mistake) and your RAM card is not being recognized by AppleWorks, Advanced Visicalc, Flashcalc, Supercalc 3A, or other programs, and you want RamWorks II.



512K Expander

2 MEG Expander

No problem. The memory chips on the card that you now have, which is where most of the money is, can be unplugged and then plugged into the expansion sockets on RamWorks II.

It's Got It All.

- 15 Day Money Back Guarantee
- Super sharp 80 column text (patent pending) with or without RGB option
- Double high resolution graphics (with or without RGB option)
- Expandable up to 1 Meg (1024K) on main board
- Expandable to 3 Meg (3072K) with expander (piggyback) card
- Can use 64K or 256K RAMS in any combination
- Linear addressing coprocessor port
- Automatic AppleWorks expansion up to 2277K desktop
- Accelerates AppleWorks
- Built-in AppleWorks printer buffer
- The only large RAM card that's 100% compatible with all IIe software
- RamDrive™ the ultimate disk emulation software included free
- 16 Bit option
- Compatible RGB option
- Built-in self diagnostics software
- No slot 1 interference
- Lowest power consumption (patent pending)
- Takes only one slot (auxiliary)
- Software industry standard
- Advanced Computer Aided Design
- Used by Apple Computer, Steve Wozniak and virtually all software companies
- Displays date and time on the AppleWorks screen with any PRO-DOS compatible clock
- 5 Year no hassle warranty

RamWorks II with 64K	**$ 179**
RamWorks II with 256K	**$ 219**
RamWorks II with 512K	**$ 269**
RamWorks II with 1 MEG	**$ 389**
RamWorks II with 1.5 MEG	**$ 549**
RamWorks II with 3 MEG	**$1699**
RGB Option (may add later)	**$ 129**
16 Bit Option (may add later)	**$ 89**

RamWorks II. The industry standard for memory expansion of the Apple IIe.
ORDER YOUR RamWorks II TODAY.
9 a.m. to 11 p.m. 7 Days, or send check or money order to Applied Engineering.
MasterCard, Visa and C.O.D. welcome. Texas Residents add 5⅛% sales tax. Add $10.00 outside U.S.A.

Applied Engineering
P.O. Box 798, Carrollton, TX 75006
(214) 241-6060

S-C Macro Assembler Quick Reference Booklet

We have a new Quick Reference Booklet for the S-C Macro Assembler! With all the new features of the Version 2.0 Macro Assemblers, including the 65C02 and 65816 in both DOS and ProDOS, we have outgrown the old Programmer Reference Card. Taking its place is our new Programmer Reference, a 14-page booklet containing even more information on the S-C Macro Assembler, even more information on the 6502/65C02/65802/65816 processors, and even more information on the Apple II, II+, //e, and //c computers.

All this new reference information is organized into an easy-to-read 14-page booklet, with the S-C Macro Assembler commands at the beginning and the 65XXX opcode tables in the center spread, so it will be as easy as possible to flip right to those important items.

These are the major subject headings covered in the new Programmer Reference:

- S-C Macro Assembler Commands
- Shorthand Commands
- EDIT Mode Commands
- DOS Commands Relevant to S-C Macro Assembler
- ProDOS Commands Active under S-C Macro Assembler
- S-C Macro Assembler Directives
- Operand Expressions
- 6502/65C02 Instructions with Opcode and Execution Cycles
- 65802/65816 Instructions
- Status Registers
- Interrupt Vectors
- Page Three Locations
- Apple Monitor Commands
- Apple Monitor Entry Points
- S-C Macro Assembler Memory Maps
- Source File Formats
- S-C Macro Assembler Parameters
- Sweet-16 Opcodes
- //e and //c Bank Switches
- ASCII Chart
- Apple II I/O Addresses

As you see, we've packed just about all of the important assembler, processor and computer information you need into this convenient 5 1/2 x 8 1/2 inch package.

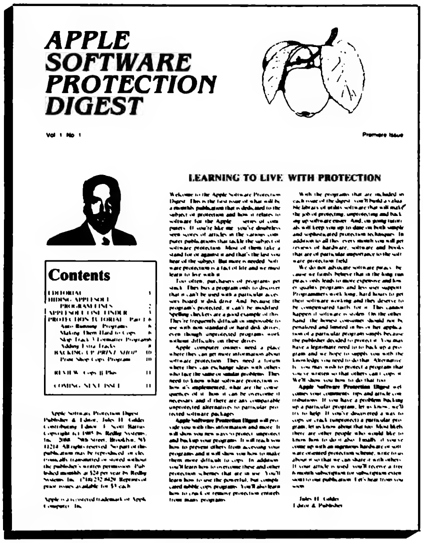
The new S-C Macro Assembler Programmer Reference is only $3.00 (plus $1.00 postage for foreign orders).

SOFTWARE PROTECTION TECHNIQUES EXPOSED!

Now, for the first time, owners of Apple // series computers can learn all about the tricks and techniques used to protect Apple software. Apple Software Protection Digest, a new monthly publication, will show you how to protect, unprotect and backup your software.

- Prevent others from accessing your programs
- Make your programs difficult to copy
- Overcome protection schemes on commercial software
- Build a library of protection-oriented utility programs
- Get help with your specific problems
- Learn about the latest advances in protection hardware and software

All this and more can be yours by subscribing to the Apple Software Protection Digest. A one-year subscription is $24, two years is $42.


APPLE SOFTWARE PROTECTION DIGEST

Vol 1 No 1

Premiere Issue

LEARNING TO LIVE WITH PROTECTION

Contents

SUBSCRIBE TODAY!

REDLIG SYSTEMS, INC., Dept. A135
2068 - 79th St., Brooklyn, NY 11214

Please enter my ________ year subscription to Apple Software Protection Digest.

☐ Enclosed is my check for ____________

☐ Please charge my credit card: ☐ VISA ☐ MasterCard ☐ American Express

Card Number ____________________ Exp. Date ______________________________ Signature ____________________

Name ____

Address ____

City ______________________________ State ________ Zip ____________________

Using Pseudo-Variables in Machine Language...........John Oakey
227 Creekstone Bend, Peachtree City, GA

A couple of years ago I got a bright idea. I was working on an Applesoft program that required knowing what files were on the disk in a given disk drive. By creating binary "images" of Applesoft variables, I was able to hook into DOS 3.3 and employ Applesoft routines to convert the information DOS 3.3 prints to the screen into regular Applesoft variables.

The whole thing worked beautifully and was printed in the last of only four "Second Grade Chats" ever published in Softalk Magazine -- in the very last issue. ("Sorree -- your number has been dis-co-nected.") I never did get paid. ($!#%~&*)

Oh, well. We Apple owners mainly do it for the love of the little machine anyway. Right? Since that time I have realized that the most important thing which I did in that article was to discover the technique of creating pseudo-variables for use in an applications program which can make available all the subroutines already written in the Applesoft ROMs.

It doesn't require a long explanation. Just one example should be enough, and it so happens that one is printed below. This short program, when called from an Applesoft program, will "poll" an Applied Engineering TIMEMASTER H.O. card from 80-column mode without affecting the screen and move the ASCII string which the time card places in the input buffer into the Applesoft variable TIME$. It not only makes getting the time while in 80-column mode possible without blowing away the screen, but it also is a great deal faster than trying to use an Applesoft interface.

This routine should also work with ThunderClock and other compatible clock cards. Permission is granted to reprint this article and to use the copyrighted program below for non-commercial applications. Have a good TIME$!

```
                  1000 *SAVE S.READ.TIME
                  1010 *--------------------------------
                  1020 *   READ TIMEMASTER H.O. CARD, PUTTING
                  1030 *       TIME INTO APPLESOFT STRING TI$.
                  1040 *--------------------------------
                  1050 *   ORIGINAL BY JOHN OAKEY, 11-22-85
                  1060 *            (c) 1985
                  1070 *
                  1080 *   MODIFIED BY BOB SANDER-CEDERLOF
                  1090 *--------------------------------
85-               1100 FORPNT .EQ $85,86
B8-               1110 TXTPTR .EQ $B8,B9
                  1120 *--------------------------------
0200-             1130 WBUF   .EQ $200
                  1140 *--------------------------------
05-               1150 SLOT   .EQ 5      <<<BE SURE TO PUT YOUR SLOT HERE>>>
                  1160 *--------------------------------
D539-             1170 AS.GDBUFS  .EQ $D539     MARK END, CLEAR HI-BITS
DA9A-             1180 AS.SAVD    .EQ $DA9A     FINISH INSTALLING STRING
DFE3-             1190 AS.PTRGET  .EQ $DFE3     PARSE STRING NAME
E3E7-             1200 AS.STRLIT  .EQ $E3E7     BUILD STRING DESCRIPTOR
                  1210 *--------------------------------
                  1220        .OR $300     (WHERE ELSE!)
                  1230 *--------------------------------
0300- A5 B8       1240 RDTIME LDA TXTPTR    SAVE CURRENT TEXT PNTR
0302- 48          1250        PHA
0303- A5 B9       1260        LDA TXTPTR+1
0305- 48          1270        PHA
```

```
                *---READ TIME INTO BUFFER--------
0306- A9 A5             LDA #"%"     MODE: "FRI JAN 03 10:11:32 AM"
0308- 20 0B C5          JSR SLOT*256+$C00B
030B- 20 08 C5          JSR SLOT*256+$C008  READ TIME STRING
                *---PREPARE STRING FOR A/S-------
030E- A2 17             LDX #23      LENGTH OF STRING
0310- 20 39 D5          JSR AS.GDBUFS     CLEAR HI-BITS AND MARK END
                *---SETUP TI$ VARIABLE----------
0313- A9 33             LDA #VARNAM
0315- 85 B8             STA TXTPTR
0317- A9 03             LDA /VARNAM
0319- 85 B9             STA TXTPTR+1
031B- 20 E3 DF          JSR AS.PTRGET
031E- 85 85             STA FORPNT
0320- 84 86             STY FORPNT+1
                *---MOVE TIME INTO TI$----------
0322- A9 01             LDA #WBUF+1  SKIP OVER LEADING QUOTE
0324- A0 02             LDY /WBUF+1
0326- 20 E7 E3          JSR AS.STRLIT
0329- 20 9A DA          JSR AS.SAVD
                *---RESTORE TXTPTR, RETURN-------
032C- 68                PLA
032D- 85 B9             STA TXTPTR+1
032F- 68                PLA
0330- 85 B8             STA TXTPTR
0332- 60                RTS
                *--------------------------------
0333- 54 49 24  VARNAM .AS /TI$/
```

And, as usual, Bob couldn't resist squeezing out a few bytes:

```
                *SAVE S.READ.TIME+
                *--------------------------------
                *   READ TIMEMASTER H.O. CARD, PUTTING
                *       TIME INTO APPLESOFT STRING TI$.
                *--------------------------------
                *   BY BOB SANDER-CEDERLOF
                *--------------------------------
81-             VARNAM .EQ $81,82
85-             FORPNT .EQ $85,86
                *--------------------------------
0200-           WBUF   .EQ $200
                *--------------------------------
05-             SLOT   .EQ 5       <<<BE SURE TO PUT YOUR SLOT HERE>>>
                *--------------------------------
D539-           AS.GDBUFS  .EQ $D539     MARK END, CLEAR HI-BITS
DA9A-           AS.SAVD    .EQ $DA9A     FINISH INSTALLING STRING
E04F-           AS.PTRGET9 .EQ $E04F     FIND OR MAKE VARIABLE
E3E7-           AS.STRLIT  .EQ $E3E7     BUILD STRING DESCRIPTOR
                *--------------------------------
                       .OR $300     (WHERE ELSE!)
                *--------------------------------
                RDTIME
0300- A9 A5             LDA #"%"     MODE: "FRI JAN 03 10:11:32 AM"
0302- 20 0B C5          JSR SLOT*256+$C00B
0305- 20 08 C5          JSR SLOT*256+$C008  READ TIME STRING
                *---PREPARE STRING FOR A/S-------
0308- A2 17             LDX #23      LENGTH OF STRING
030A- 20 39 D5          JSR AS.GDBUFS     CLEAR HI-BITS AND MARK END
                *---SETUP TI$ VARIABLE----------
030D- A9 54             LDA #'T'     HI-BIT OFF FOR STRING VARIABLE
030F- 85 81             STA VARNAM
0311- A9 C9             LDA #"I"     HI-BIT ON FOR STRING VARIABLE
0313- 85 82             STA VARNAM+1
0315- 20 4F E0          JSR AS.PTRGET9
0318- 85 85             STA FORPNT
031A- 84 86             STY FORPNT+1
                *---MOVE TIME INTO TI$----------
031C- A9 01             LDA #WBUF+1  SKIP OVER LEADING QUOTE
031E- A0 02             LDY /WBUF+1
0320- 20 E7 E3          JSR AS.STRLIT
0323- 4C 9A DA          JMP AS.SAVD  CLEAN UP STRINGS AND RETURN
                *--------------------------------
```

Computing the Day of Week..............Bob Sander-Cederlof

Within reasonable limits, it should be possible for a clock/calendar card to automatically set the day-of-week number when given the year, month, and day. The algorithm for deriving day-of-week from the date is simple enough. However, as the algorithm is stated in all my reference material, it involves multiplication and division by numbers that are not simple powers of two.

I have simplified the algorithm so that it will work over the range from March 1, 1984 through December 31, 2083. That should be an adequate range for any Apple products!

Years evenly divisible by 4 are leap years, having 366 days. The years ending in 00 are exceptions, unless they are divisible by 400. Thus 1900 was not a leap year, 2100 will not be a leap year, but 2000 is a leap year.

My algorithm started out as a method for converting a Y-M-D date to a Julian date, which is a unique number that was 0 several thousand years ago. I could get the remainder after dividing the Julian date by 7, and use it for a day-of-week index. However, the numbers get rather large; they won't fit in one byte.

By converting all the intermediate values to their modulo 7 equivalents, I can keep the result down to byte-size. Here is an Applesoft program which implements my algorithm:

```
100 DIM MD(11),D$(6)
110 DATA 3,6,1,4,6,2,5,0,3,5,1,4
120 DATA SUN,MON,TUES,WEDNES,THURS,FRI,SATUR
130 FOR I=0 TO 6 : READ M : MD(I)=M : NEXT
140 FOR I=0 TO 11: READ D$: D$(I)=D$: NEXT
200 INPUT Y,M,D
210 M = M-3
220 IF M<0 THEN M=M+12 : Y=Y-1
230 Y=Y-1984
240 W = Y + INT(Y/4) + MD(M) + D
250 IF W>6 THEN W=W-7 : GO TO 250
260 PRINT D$(W)"DAY"
270 GO TO 200
```

Lines 100-140 build two arrays. The MD array holds a modulo 7 number for the number of days preceding each month in a normal year (not leap year). The D$ array holds the names of the days, shortened by the last three letters.

Line 200 waits for you to type in the year, month, and day as three numbers. I did not add any error testing, but I expect the year to be from 1984 up. The month should be a number from 1 to 12, and the day from 1 to 31.

Lines 210-220 adjust the month number. I move January and February to the end of the previous year, like it must have been in the olden days. That makes leap day the last day of the year, where it belongs. It also makes the month names for

Sept-, Oct-, Nov-, and Dec-ember make linguistic sense! March becomes the first month, December the tenth, and so on. Internally, the value of the variable M will be a number from 0 to 11.

Line 230 adjusts the year to start at 1984. Line 240 adds up the various day-values. We add Y, the number of the years since 1984, because 365 = 1 mod 7. We add INT(Y/4) to get the leap days. MD(M) adds in the bias for the number of days beyond an integral number of weeks to the end of the previous month. D adds in the day number. Altogether we have a number which is still less than 256, and fits in one byte in a machine language version of the algorithm.

Line 250 subtracts 7 (whole weeks) until we get to a number less than 7. The result is the day number in a week with 0 meaning Sunday, 1 meaning Monday, and so on. Line 260 prints the day name, and line 270 lets us try another date.

After making sure of my method with the Applesoft program, I coded it in assembly language. The program which follows is set up to be used from inside Applesoft, and I also list here the Applesoft driver. I did it this way to make it easy to test my assembly language code. Later I will probably put the code inside a larger package which sets the time and day on my clock card. Once it is in there, I can forever forget about the need to tell the card what day of week it is.

```
                  1000 *SAVE S.DAY OF WEEK
                  1001        .OR $300
                  1010 *--------------------------------
0300-             1020 YEAR   .BS 1        84-99 MEANS 1984-1999; 0-83 MEANS
0301-             1030 MONTH  .BS 1        1...12 FOR JAN...DEC    2000-2083
0302-             1040 DAY    .BS 1        1...31
0303-             1050 W      .BS 1
                  1060 *--------------------------------
                  1070 DOW
0304- AD 00 03    1080        LDA YEAR     NORMALIZE YEAR TO 1984
0307- 38          1090        SEC          SO IT RUNS 1...99
0308- E9 54       1100        SBC #84      (MAR 1, 1984 THROUGH DEC 31, 2083)
030A- B0 02       1110        BCS .1       WAS 1984-1999
030C- 69 64       1120        ADC #100     WAS 2000-2083
030E- 8D 03 03    1130 .1     STA W
                  1140 *--------------------------------
0311- AD 01 03    1150        LDA MONTH    ADJUST MONTH SO FEBRUARY IS END
0314- 38          1160        SEC                                   OF YEAR
0315- E9 03       1170        SBC #3
0317- B0 05       1180        BCS .2
0319- CE 03 03    1190        DEC W
031C- 69 0C       1200        ADC #12
031E- AA          1210 .2     TAX
                  1220 *--------------------------------
031F- AD 03 03    1230        LDA W        YEAR
0322- 4A          1240        LSR
0323- 4A          1250        LSR
0324- 18          1260        CLC          + INT (YEAR/4)
0325- 6D 03 03    1270        ADC W
0328- 7D 39 03    1280        ADC MD,X     + MD(ADJ.MONTH)
032B- 6D 02 03    1290        ADC DAY      + DAY
                  1300 *--------------------------------
032E- 38          1310        SEC
032F- E9 07       1320 .3     SBC #7       MOD 7
0331- B0 FC       1330        BCS .3
0333- 69 07       1340        ADC #7
0335- 8D 03 03    1350        STA W
0338- 60          1360        RTS
```

```
                          1370 *--------------------------------
0339- 03 06 01
033C- 04 06 02
033F- 05 00 03
0342- 05 01 04 1380 MD      .DA #3,#6,#1,#4,#6,#2,#5,#0,#3,#5,#1,#4
                          1390 *--------------------------------
```

Lines 1020-1050 are the variables used to communicate with the Applesoft test program, by way of PEEKs and POKEs. The program assumes that only the last two digits of the year are used, so that YEAR is a number from 84 to 99 for 1984 to 1999; values from 0 to 83 signify years from 2000 to 2083.

Lines 1080-1130 change the year number, which runs 84...99 and 00...83 to a value based at 1984, running from 00 to 99. 00 means 1984, 99 means 2083.

Lines 1150-1210 are equivalent to the Applesoft lines 210 and 220 in the first program above. Lines 1220-1290 are equivalent to the Applesoft line 240. Lines 1300-1340 reduce the result to a modulo 7 remainder. The final value, a number from 0 to 6, is stored in line 1350 where an Applesoft driver can find it by PEEK(771).

Here is my Applesoft test program. This time I went in for a little range checking on the input values for year, month, and day.

```
100 DIM D$(6)
110 DATA SUN,MON,TUES,WEDNES,THURS,FRI,SATUR
120 FOR I=0 TO 6 : READ M : MD(I)=M : NEXT
200 INPUT "YEAR (1984-2083):  ";Y
210 IF Y<1984 OR Y>2083 THEN 200
220 Y = Y - INT(Y/100)*100
230 POKE 768,Y
300 INPUT "     MONTH (1-12):  ";M
310 IF M<1 OR M>12 THEN 300
320 POKE 769,M
400 INPUT "       DAY (1-31):  ";D
410 IF D<1 OR D>31 THEN 400
420 POKE 770,D
500 CALL 772
510 W = PEEK(771)
600 PRINT D$(W)"DAY"
610 GO TO 200
```

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).